A Web-based evolutionary model for Internet Data Caching

Athena Vakali
Department of Informatics
Aristotle University
54006 Thessaloniki, Greece
email: {*avakali*}*@csd.auth.gr*

Abstract

Caching is a standard solution to the problem of insufficient bandwidth caused by the rapid increase of information circulation across the Internet. Cache consistency mechanisms are a crucial component of each cache scheme influencing the cache usefulness and reliability. This paper presents a model for optimizing Internet cache content by the use of a genetic algorithm and examines the model by trace-driven experiments. Cached data are considered as a population evolving over simulated time by a number of successive cache "generations". The model is tested by the use of traces provided by a Squid proxy cache server. Using trace-driven caching, we show that the proposed evolutionary mechanisms improve cache non-staleness and consistency and result in an updated cache content.

Index terms: *Internet applications, Web-based object caching, evolutionary computation algorithms.*

1 Introduction

Internet is a widely-expanded distributed system with numerous servers which maintain caches. Improving Web cache content is a major issue for facilitating and fastening the information availability and reliability. World-Wide Web information circulation has been almost doubling every six months, and despite efforts for capacity increases demands aren't always kept up [8]. Caching presents an effective solution, since it provides mechanisms to faster web access and improves load balancing without demanding more bandwidth. The *World-Wide Web caching* differs from a distributed file system mainly in its access patterns since Web is orders of magnitude larger than any distributed file system [7, 3]. *Intelligent Caching* has been investigated in [12] whereas the potential for document caching at the application-level has been tested in [2]. Two other factors affecting cached data management are the *Caching and replication* [1] and the *Cache hierarchy* [4, 9].

The most significant problem of a cache server is its ability to keep data as "fresh" as possible i.e., eliminate Web information staleness. The Web server is responsible for the preservation of information object's "freshness" towards balancing the server load and facilitating the internet information circulation. *Cache consistency* policies have been included in almost every proxy cache server (e.g. [9]) and their improvement became a major research issue. In [7] a survey of contemporary cache consistency mechanisms in Internet is presented and examines recent research in Web cache consistency.

Evolutionary strategies have been used to solve many computational problems demanding optimization and adaptation to changing environments. Usually grouped under the term evolutionary algorithms or evolutionary computation, are the domains of Genetic Algorithms, evolution strategies, and genetic programming. More specifically, Genetic Algorithms have been applied in various research areas such as scientific modeling, machine learning as well as network infrastructure [6, 11, 5].

This paper presents a model which adapts the evolutionary computation idea to preserve a consistent cache 'population' of World-Wide Web information objects. A Web cache is modeled as a population of information objects and the aim is to improve the cache population regarding its consistency, availability and accessibility. The model is based on the Squid proxy-cache server environment and is experimented under real cache traces and cache log files. The remainder of the paper is organized as follows. The next section describes proxy cache environments and presents the Genetic Algorithm process. Sections 3 and 4 present the implementation model and the trace driven experimentation, respectively. Section 5 points some conclusions and discusses potential future work.

2 The Internet Caching model

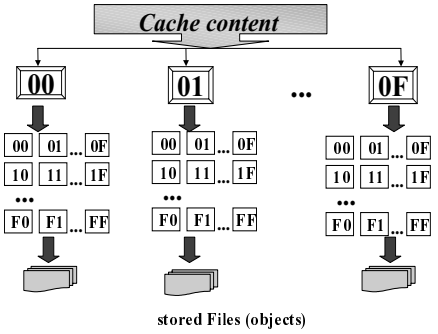

Figure 1: Squid proxy cache structure.

Caching was initially introduced to provide an intermediate storage space between the main memory and the processor and the caching principle was extended to Web servers by considering them as another level in the memory hierarchy.

2.1 Caching on the Web

A Web cache is an application residing between Web servers and clients and watches requests for information objects identified as html pages, images and files saving a copy for itself. If another request concerns the same object, cache will use the copy it has, instead of asking the original server for it again. The two main Web caches advantages are the reduce in both latency (request is satisfied by the cache which is closer to the client) and traffic (each object is gotten from the server once, thus reducing the bandwidth used by a client). Nowadays, a variety of cache servers are available for the World-Wide Web caching and most of the recent Web servers include caching modules (e.g. Apache, Spinner, Jigsaw, Purveyor).

- *CERN proxy server* has been widely adopted since there was a large infrastructure of the CERN web servers already installed. A heuristic known as *time-to-live* (TTL), was used to manage object's staleness. A TTL timing frame based on request and expiry dates, accompanies each document in the cache [12, 7].
- *Netscape Proxy Server* has been available commercially since 1995 and supports management of object's staleness by TTL frames based on object's age when it is cached.
- *Harvest cache* software was developed with the aim of making effective use of the information available on the Internet, by sharing the load of information gathering and publishing between many servers. The first version of the Harvest cache software supported TTLs, whereas more recent Harvest versions support new options for the management of object's staleness. Newest Harvest developments are available commercially whereas a team from the National Laboratory for Advanced Networking Research (NLANR) have continued to provide a free version under the name Squid [10].

Squid caching software has gained a lot of attention lately, since it is used on an experimental network of seven major co-operating servers across U.S.A. This network is established under a project framework by the National Laboratory for Advanced Networking Research (NLANR) and supports links to collaborating cache projects in other countries. Aristotle University has installed Squid proxy cache for main and sibling caches and supports a Squid mirror site. The present paper uses for experimentation traced information provided by this cache installation . Figure 1 represents the organization of cache hierarchy structure which consists of a two-level structure. Assuming approximately 256 objects per directory there is a potential of a total of 1,048,576 (=16 $\times$256$\times$256) cached objects.

2.2 Evolutionary computation - Genetic algorithms

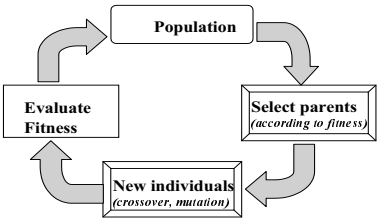

Figure 2: The Genetic Algorithm process.

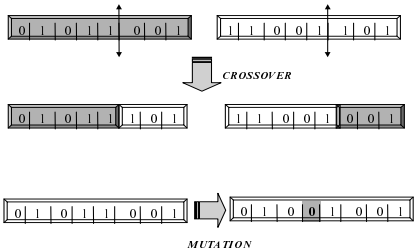

Figure 3: operators: crossover and mutation.

Evolutionary programming has been successfully applied to numerous problems from different domains. Genetic algorithms (*GAs*) comprise one of the main evolutionary methods, applied to many computational problems requiring either search through a huge number of possibilities for solutions, or adaptation to a changing environment. A GA is an iterative proce-

dure that consists of a constant-size population of individuals each one represented by a finite string of symbols, encoding a possible solution in a given problem space. The standard GA generates an initial population of individuals, which is updated at each evolutionary step resulting in a new "generation". The individuals in the current population are decoded and evaluated according to some predefined quality criterion, called *fitness function*. Figure 2 depicts the cycle of a GA applied in a space of individuals. Each individual's fitness evaluation is an important parameter, usually given as part of the problem's description. Two genetically-inspired operations, known as *crossover* and *mutation* are applied to selected individuals in order to successively create stronger generations. Figure 3 depicts these two operations in a 8-bit string individual.

Crossover is performed between two individuals (parents) with some probability, in order to identify two new individuals resulting by exchanging parts of parents' strings. The exchanging of parents parts are performed by cutting each individual at a specific bit position and produce two "head" and two "tail" segments. The tail segments are then swapped over to produce two new full length individual strings.

Mutation is introduced in order to prevent premature convergence to local optima by randomly sampling new points in the search space. Mutation is applied to each child individually after crossover. It randomly alters each individual with a (usually) small probability (e.g. 0.001).

Provided that GA has been correctly implemented, the population will evolve over successive generations such that the fitness of the best and the average individual in each generation is improved towards the global optimum.

3 The GA Cache model

The presented implementation adapts a GA evolutionary approach to the cache information objects update in order to optimize cacheable objects and result in an improved cache content. The GA is used because of two main reasons : First, the basic idea of the GAs is based on the evolution of populations by the criterion "survival of the fittest" and the the cache should contain the fittest (most fresh) information objects. Second, the GAs are applied to problems demanding optimization out of spaces which are too large to be exhaustively searched and the cache content consists of million(s) of information objects (stored files) as indicated in Subsection 2.1. The proposed GA model follows the Simple GA proposed in [6].

3.1 The Cache update algorithm

The following heuristics were made in order to adapt the GA approach to the cache update scheme:

- cache is considered as a population of individuals,
- the individual is the actual cached object, identified by the filename where it's stored. Squid objects, the last level in the cache storage hierarchy (Figure 1) are stored in files with filenames coded as hexadecimal numbers strings (e.g. 001af200, 000be301)
- each individual is assigned with a fitness value derived by a general fitness function declaring the object's freshness.

The cache update scheme is implemented in order to perform cache reform and re-"generation" at regular time intervals. A pseudo-code version of the GA scheme of our cache-update implementation follows:

```
initialize()    // old_pop=initial population
generation <- 1
while (generation <= maxgen)  do
   par1 <- selection(popsize, fitness, old_pop)
   par2 <- selection(popsize, fitness, old_pop)
   crossover(par1,par2,old_pop,new_pop,p_cross)
   mutation(new_pop, p_mutate)
   statistical_report(new_pop)
   old_pop <- new_pop
   generation <- generation + 1
```

In the above GA *maxgen* corresponds to the maximum number of successive generation runs, *popsize* is the cache population size, $fitness$ is the cache update factor (fitness evaluation is described in the next subsection), $par1$ and $par2$ are the parents chosen for the reform of each generation, p_cross, p_mutate are the probabilities for crossover and mutation, respectively. Furthermore, each individual is decoded as a binary bit string corresponding to the hexadecimal string of the object's filename. The implementation of crossover above gets old_population and results in new_population either by preserving the parents or by reproducing a new individual based on parents fitness. Mutation is performed on the new_population by affecting few positions in the individuals string towards a better fitness.

3.2 GA Cache Metrics

As mentioned earlier, cache generations are evolved based on specific fitness function related to the object's freshness/staleness factor. In order to identify this factor, Squid's log files fields are used for each object's fitness function. Squid (in its default configuration) makes four logfiles:

- *logs/access.log*: requests issued to the proxy server regarding how many users use the cache, how much each requested etc.

- *logs/cache.log*: information Squid needs to know such as errors, startup messages etc.
- *logs/store.log*: information of what's happening with our cache diskwise; it shows whenever an object is added or removed from disk.
- *cache/log*: contains the mapping of objects to their disk location.

Store log fields	
time	time this entry was logged.
action	RELEASE, SWAPIN, or SWAPOUT. RELEASE : object removed from cache. SWAPOUT : object saved to disk. SWAPIN : object swapped into memory.
status	HTTP reply code.
datehdr	HTTP Date: reply header.
lastmod	HTTP Last-Modified: reply header.
expires	HTTP Expires: reply header.
type	HTTP Content-Type reply header.
exp-len	HTTP Content-Length reply header.
real-len	# bytes of content actually read.
method	HTTP request method.
key	cache key ; often simply the URL.

Table 1: *store.log* :Fields of each individual object

Since fitness function drives the evolution of the GA population, is important to reward the improved cache content individuals. Therefore, a metric characterizing cache object's freshness will be the best choice for the GA caching scheme. As described in Section 2 all proxy caches relate their object's refresh policy with timing object's last modification period. Therefore, in our GA caching, each individual object's fitness is evaluated by a factor corresponding to the ratio of object's "ages" since retrieval and its last modification. More specifically,

$$fitness_{object} = \frac{age1}{age2}$$

where $age1$ corresponds to the time that passed since the object's retrieval and $age2$ is the age of the object at the time of its retrieval. Fields of *store.log* (described in Table 1) are used to evaluate each of these parameters. By using these tags the evaluation of the numerator results in $age1 = now - datehdr$ and the denominator in and $age2 = datehdr - lastmod$.

4 Experiments - Results

The simulator was tested and validated by Squid cache traces and their corresponding log files. Traces refer

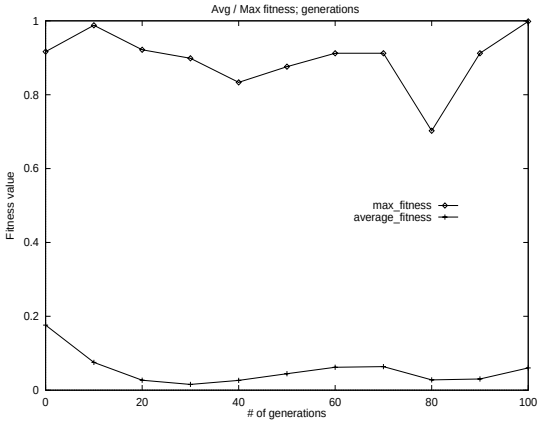

Figure 4: avg/max fitness over generations

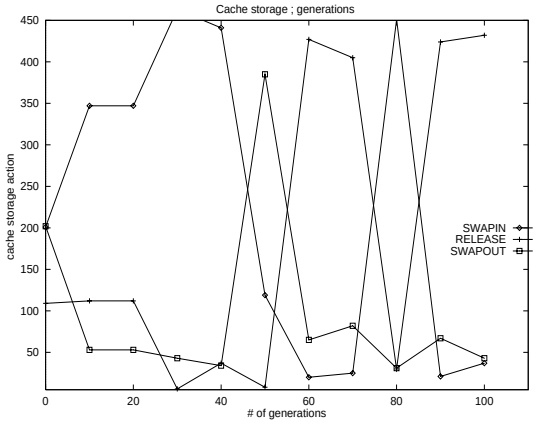

Figure 5: actions over generations

to the period from October to December 1998 and information objects refer to a typical 5-day run over the overall trace period. GA scheme was applied to cache population when hits were reduced, i.e. at night simulated time. Initial population for the GA scheme is the population produced by Squid proxy. Figures 4, 5, 6 and 7 depict the effect of the number of generations to the cache metrics. More specifically, Figure 4 presents the average and maximum fitness values for a cache population being reproduced by $10, 20, \cdots, 100$ generations. Crossover probability is 0.6 whereas mutation probability is 0.0333, since these values have been suggested as a representative trial set for most GA optimizations. The population's average fitness value decreases, even for the 10 generations reproduction, whereas the maximum fitness remains at almost the same rate. Decreasing average objects fitness, i.e. last modified ratios is important since cache will consist of more "fresh" members. The decreasing rate in the last modified ratios (fitness) is more than 50% for some generations. Figure 5 presents the actions (SWAPIN, RELEASE, SWAPOUT) that each cache generation will take. It is important to note that as

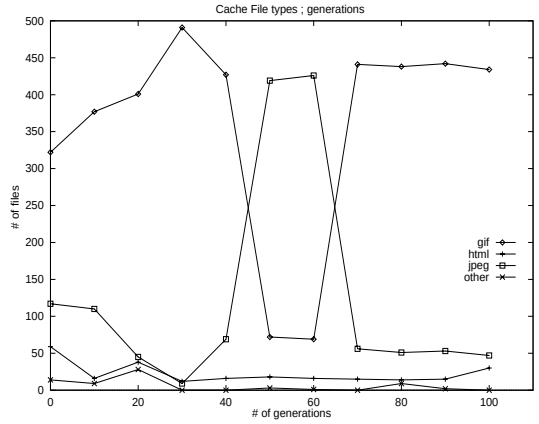

Figure 6: file types over generations

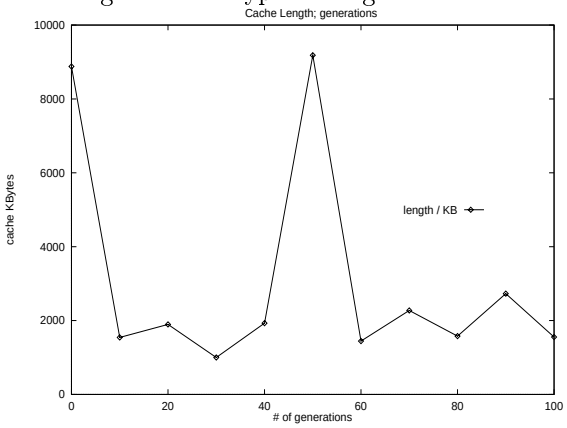

Figure 7: KBytes over generations

generations evolve, the GA scheme attempts to favor the RELEASE action in order to update cache. Figure 6 represents the file types content of each cache generation. Files include the *html, gif, jpeg* types which are the most common in cache populations and all *other* types include mostly plain/text files as well as application files. The *gif* and *jpeg* files are the most commonly used in each generation, whereas *html*s remain at almost the same rate. Figure 7 refers to the KBytes length variation at each cache generation. This length is the actual length (*real_len*) and not the expected length (*exp_len*) of the store.log headers (Table 1). This length is reduced significantly for larger number of generations (>50) and depends on the file types that cache contains, since more *gif* and *jpeg* files occupy more space.

5 Conclusions - Future Work

The Internet data caching is studied under a GA model for preserving cache consistency. The simulation process included almost all of the necessary parameters to study the model under real Squid cache traces, with support to the most indicative cache parameters (last modification factor, cache length, actions and file types). The GA scheme has been proven quite effective since cache population evolved over the simulation time for an increasing number of generations.

Further research should further experiment the present scheme under different fitness selection policies. Other evolving computation schemes (e.g. simulated annealing, threshold acceptance), could be adopted in internet caching, in order to study their effect on cache consistency.

References

[1] M. Baentsch et al.: Enhancing the Web's Infrastructure: From Caching to Replication, *IEEE Internet Computing*, Vol.1, No.2, pp. 18-27, Mar-Apr 1997.

[2] A. Bestavros, R.L. Carter and M. Crovella: Application-level Document Caching in the Internet, *Proceedings of 2nd International Workshop in Distributed and Networked Environments*, SDNE 1995.

[3] M. A. Blaze: Caching in Large-Scale Distributed File Systems, Princeton University, PhD thesis, Jan 1993.

[4] A. Chankhunthod, P. Danzig and C. Neerdaels: A Hierarchical Internet Object Cache, *Proceedings of the USENIX 1996 Annual Technical Conference*, pp.153-163, San Diego, California, Jan 1996.

[5] B. Dengiz, F. Atiparmak, A. E. Smith : Local Search Genetic Algorithm for Optimization of Highly Reliable Communications Networks, *IEEE Transactions on Evolutionary Computation*, Vol.1, No. 3, pp. 179-188, Aug 1997.

[6] D. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[7] J. Gwertzman and M. Seltzer: World Wide Web Cache Consistency, *Proceedings of the USENIX 1996 Annual Technical Conference*, pp.141-151, San Diego, California, Jan 1996.

[8] A. S. Heddaya: DynaCache: Weaving Caching into the Internet, Infolibria, 1998.

[9] O. Pearson: The Squid Cache software, Squid Users Guide, http://www.auth.gr/SquidUsers/, 1998.

[10] Squid: Squid Internet Object Cache, http://www.auth.gr/Squid/, 1998.

[11] T. Starkweather, D. Whitley and K. Mathias: Optimization Using Distributed Genetic Algorithms, *Parallel Problem Solving*, Springer Verlag, 1991.

[12] D. Wessels: Intelligent Caching World-Wide Web Objects, *Proceedings of the INET'95 Conference*, Jan 1995.